Project 1 – A Perpetual Calendar

The goal of this assignment is to write a simple program that lets the user enter a date and have it report back the day of the week that the date fell on.

Documentation and Style

Project 1 is worth **100 points**. When you submit Project 1, Web-CAT will test that your program works correctly and creates the proper output, if your program works correctly you will receive **80 points** from the automated testing provided by Web-CAT. The remaining **20** points are for style and documentation, which will be **manually graded by the TAs**, using the rubric posted on the course website.

Background

A frequent need in many computer applications is to be able to compute the day of the week that a Gregorian calendar date falls on, given a month, day, and year. Most programming languages include functions to perform these calculations already, but you will write the calculations yourself to see how basic arithmetic is performed in Python.

One method for computing the day of the week of a calendar date is known as **Zeller's congruence,** devised by German mathematician Christian Zeller (1822-1899). For those interested in the (math-heavy) derivation of his formula, you can see the Wikipedia article, Zeller's congruence. However, all of the information that you will need for this project is presented below as well.

Zeller's congruence is a straightforward mathematical formula, but there is one important point that you must be aware of. Zeller recognized that the leap day February 29th would complicate the calculation since it falls in the middle of the year. To simplify things, he "moved" the leap day to the end of the year by treating January and February not as the first and second months of year Y, but as the 13th and 14th months of the year Y – 1. **For example, if you wanted to find the day of the week for February 2nd, 2009, you would need to perform the calculation for the 2nd day of the 14th month of the year 2008.**

The following is the first part of Zeller's congruence:

$$Z = d + \left\lfloor \frac{26(m_z + 1)}{10} \right\rfloor + y_z + \left\lfloor \frac{y_z}{4} \right\rfloor + 6\left\lfloor \frac{y_z}{100} \right\rfloor + \left\lfloor \frac{y_z}{400} \right\rfloor$$

Where

- d is the day of the month,
- m_z is the month, modified to put January and February at the end of the year (3 = March, 4 = April, ..., 12 = December, 13 = January, 14 = February),
- y_z is the year, modified by subtracting 1 if the month was January or February,
- and Z is the result so far.

The $\lfloor \; \rfloor$ symbols are the mathematical "floor" function; it means round down whatever is inside them to the nearest integer. In Python, you can compute $\lfloor x \rfloor$ by importing the `math` module calling the function `math.floor(x)`. Another possibility is to use **integer division (//)** throughout the equation above, rather than the standard division operator. Either option is acceptable.

Once you have the value Z above, you can convert it to a number that represents the day of the week with the following formula:

$$w = ((Z + 5) \mod 7) + 1$$

Where `w` will equal 1 for Monday, 2 for Tuesday, and so on, up to 7 for Sunday. Remember that mod is the remainder after division, represented in Python by the % operator. (For those curious, this particular numbering system corresponds to the ISO 8601 standard for date/time data, but you don't have to know this.)

As far as those of us in the United States are concerned, this formula will work for any date on or after September 14, 1752 – the day the Gregorian calendar was adopted by the British Empire and its holdings. Dates prior to that used the Julian calendar, so we won't be worrying about those. (Mind-blowing fact: What day came before September 14, 1752? It was September 2, 1752.)

Sample input and corresponding output
This program is designed to be run interactively; the user should be prompted to enter a date as a string and two integers (the name of the month, the day, and the 4-digit year, in that order) as separate prompts, and then your program will produce a message showing the day of the week.

The following is an example of what you might see when you run the program; the input you type will be shown in green (press Enter at the end of a line), and the output generated by the program is shown in black.

```
Enter the month: January
Enter the day: 29
Enter the year: 2000
Saturday
Would you like to enter another date (y/n): y
Enter the month: January
Enter the day: 30
Enter the year: 2000
Sunday
Would you like to enter another date (y/n): n
```

Notice that the **months** you input should be the human readable names, e.g. "January" rather than numbers. So before performing the equations above your program would convert "January" to 13, "February" to 14, "March" to 3, etc.

Also notice that your program should ask whether another date should be entered. If the response to the prompt is "y" your program should ask for another date, if the user inputs an "n" the program should end. At this point you don't have to worry about incorrect input, when asked "Would you like to enter another date (y/n): ", the input will always be a "y" or an "n" character.

Additionally, your program should be able to calculate **an arbitrary number of dates.** In the example above there were only 2 dates calculated, but I should be able to ask for 10, 20, 50, or more different dates during the course of running your program without issue.

Feel free to modify the prompt or the output to personalize it. But since it's being auto-graded, make sure that you follow a few simple guidelines:

- The date **must** be entered in the order month, day, and then year. This makes it easier to read the date using three easy `input()` commands.
- The program should let the user enter one date, print the day of the week, and then prompt whether to process additional dates.
- If you modify the output of the program in any way, make sure that only one day of the week shows up, ever, in the output. What this means is, **do not** write a prompt that says something like "This program will tell you the day of the week, Monday through Friday".
- Of course, make sure that the names of the days of the week you print are spelled correctly.

Plan of Attack

If you're relatively new to programming, don't try to write the entire program at once. Come up with a plan of attack, break the task down into smaller sub-problems, and test your work after each stage. Here's an example plan of attack:

1. Create your Python file (referring back to Homework 1's instructions if necessary). You **must** name the file **p1.py**, and remember **not to use spaces in the name.**
2. You will need some sort of loop. This loop will continue executing as long the user enters a "y", when asked whether they want to enter more dates. The loop will stop when the user enters "n" to the prompt.
3. Start by writing the part of the program that asks the user for the date. Print out the prompts, and read in the string and two integers. For debugging purposes, try printing the input back out to see if they came in correctly. (But delete these debugging output statements before you're done!)
4. After that, use a selection (`if`) statement to convert the month names into the appropriate numbers. If the date is in January or February, the month number and year need to be modified according to the rule above. Try printing them out again and run a few tests to see if your code works the way you expect.

5. Now, compute Zeller's congruence (both equations at the top of this write-up).
6. Use the result of Zeller's congruence, which should always be a number 1–7 if you wrote it correctly, in an `if` statement to print out the correct day of the week.

Testing Your Program
You are expected to thoroughly test your program before you submit it for grading. Make sure you try dates that give you each day of the week; try dates in different months, different years, and different centuries. Make sure you test "edge cases" as well, such as leap years or dates just before or after a millennium. Also try calculating multiple dates in a row.

Don't worry about invalid dates, however. Your program does not need to handle dates that don't exist, like January 37 2010. Also, at this point you don't have to worry about incorrect input, when asked "Would you like to enter another date (y/n): ", the input will always be a "y" or a "n".

What to Submit
For this assignment you should submit your **p1.py** file.

This assignment will be graded automatically. Test your programs thoroughly before submitting them. Make sure that your programs produce correct results for every logically valid test case you can think of. Do not waste submissions on untested code, or on code that does not compile with the supplied code from the course website.

Web-CAT will assign a score based on runtime testing of your submission; your best score will be counted; the TAs will later verify that your best submission meets the stated restrictions, and assess penalties if not.

To submit this assignment:
1. Visit http://web-cat.cs.vt.edu in your web browser.
2. Enter your Virginia Tech PID and password in the appropriate fields on the log-in screen, and make sure that **Virginia Tech** is selected as the institution. Click **Login**.
3. The Web-CAT home screen will display useful announcements and assignments that are currently accepting submissions. Find the assignment that you want to submit in the table, and click the "Submit" button next to it.
4. Click the **Browse...** button and select the file you want to upload. The homework assignments and programming projects for this course should be self-contained in a single **.py** file, so you can simply select that one file.
5. Click the **Upload Submission** button. The next page will ask you to review your selection to ensure that you have chosen the right file. If everything looks correct, click **Confirm**.

The next page will show that your assignment is currently queued for grading, with an estimated wait time. This page will refresh itself automatically, and when grading is complete you will be taken to a page with your results.

When your results are ready, make sure that you have **80%** on the assignment. If you have anything less, read the hints that Web-CAT gave you and make any corrections to your code that you need to make, then submit again. Remember that for the programming projects in this class (as opposed to the homework assignments), you can submit up to 3 days after the due date, with a 10% penalty per day late.

Pledge

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the submitted file:

```
#  <include a description of the purpose of this file/project/package>
#
#  @author <name and surname> (your VT PID)
#  @date   <the date>
#
# Virginia Tech Honor Code Pledge
#  On my honor:
#
#  - I have not discussed the Python language code in my program with
#    anyone other than my instructor or the teaching assistants
#    assigned to this course.
#  - I have not used Python language code obtained from another student,
#    or any other unauthorized source, either modified or unmodified.
#  - If any Python language code or documentation used in my program
#    was obtained from another source, such as a text book of coarse
#    notes, that has been clearly noted with a proper citation in
#    the comments of my program.
#  - I have not designed this program in such a way as to defeat or
#    interfere with the normal operation of the Web-Cat Server.
#
#  <your name>
```

Failure to include this pledge in a submission will result in the submission being disallowed during code review.